NPS-54-89-09

NAVAL POSTGRADUATE SCHOOL

Monterey, California

HOW A STANDARDIZED CHANGE MANAGEMENT METHODOLOGY CAN IMPROVE SOFTWARE MAINTENANCE

Norman F. Schneidewind

May 1989

Prepared for: Navy Management Systems Support Office
Naval Air Station
Norfolk, VA 23511-6694

NAVAL POSTGRADUATE SCHOOL
Monterey, California

RADM. R. C. Austin
Superintendent

Harrison Shull
Provost

The research summarized herein was sponsored by the Navy Management Systems Support Office under N6856187P030034.

This report was prepared by:

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; Distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S): NPS-54-89-09	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: Navy Management Systems Support Office; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N6856187P030034
8c. ADDRESS (City, State, and ZIP Code): Naval Air Atation Norfolk, VA 23511-6694	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO.; WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
How a Standardized Change Management Methodology Can Improve Software Maintenance

12. PERSONAL AUTHOR(S)
Norman F. Schneidewind

13a. TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Technical Report	FROM Aug. 88 TO May 89	1989 May 31	26

16. SUPPLEMENTARY NOTATION

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software Maintenance; Local Area Network

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this report is to assist the Navy Management Systems Support Office in performing software maintenance by showing a detailed example of applying the software change management methodology which was described in the previous report: 'Software Maintenance: The Need for Standardization', Norman F. Schneidewind, February 1989, Naval Postgraduate School Technical Report NPS-54-89-02. The maintenance of local area network software is used as the example.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code): (408) 646-2719; 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

HOW A STANDARDIZED CHANGE MANAGEMENT METHODOLOGY CAN IMPROVE SOFTWARE MAINTENANCE

by

Norman F. Schneidewind

May 1989

How a Standardized Change Management Methodology Can Improve Software Maintenance

by

Norman F. Schneidewind

May 1989

Prepared for: Navy Management Systems Support Office
Norfolk, VA 23511-6694

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract- The purpose of the this report is to assist the Navy Management Systems Support Office in performing software maintenance by showing a detailed example of applying the software change management methodology which was described in the previous report: 'Software Maintenance: The Need for Standardization', Norman F. Schneidewind, February 1989, Naval Postgraduate School Technical Report NPS-54-89-02. The maintenance of local area network software is used as the example.

I. INTRODUCTION

Software maintenance is a major activity at the Navy Management Systems Support Office (NAVMASSO). The purpose of the this report is to assist the Navy Management Systems Support Office in performing software maintenance by showing a detailed example of applying the software change management methodology which was described in the previous report: 'Software Maintenance: The Need for Standardization', Norman F. Schneidewind, February 1989, Naval Postgraduate School Technical Report NPS-54-89-02. The maintenance of local Area network software is used as the example. The methodology is general and can be applied to any programming environment and language, including COBOL.

As an introduction to the subject of software maintenance, we provide some definitions followed by an explanation of the importance of the subject.

A. Definitions

Software Maintenance: Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment [1].

This definition is the conventional one and is useful if our interest in modification to software is limited to changes that are made after the software is delivered. However, it is a fact that changes are not confined to the post-delivery phase; they are made during all life cycle phases. In some cases, changes are made in significant numbers prior to delivery.

Maintainability: The ease with which a software can be maintained [1].

Change Management: The process of making changes to software and controlling their effects during the entire life of the software.

The last definition recognizes the fact that modifications to software must be managed effectively during the entire life of the software. It is the definition used here.

According to various sources, software maintenance accounts for a significant amount of the total time and cost of running a data processing organization. For example, one study reports the following: about half of applications staff time spent on maintenance, over 40 percent of the effort in supporting an operational application system spent on user enhancements and extensions, and about half a man-year of effort allocated annually to maintain the average system (2). In another report the same authors list the factors which cause the significant maintenance effort: system age, system size, relative amount of routine debugging, and the relative development experience of the maintainers (3). System age drives the other factors: with increased system age, system size increases, leading to greater effort allocated to routine debugging, and with increased system age, the relative development experience of the maintainers declines due to organizational turnover and change. All of these factors tend to increase the time and cost of performing maintenance. Thus maintenance is an area that deserves a lot of attention. Improvements in maintenance practices should result in reduced costs and increased effectiveness of performing maintenance.

However there is a limit to reducing cost and increasing effectiveness through improved practices, because the maintainability of the software has largely been determined by the developer before it ever reaches the maintainer. The maintainer can only influence quality during the maintenance phase of the software life cycle. The quality of the software as designed is determined, in part, by whether the software development methodology assists the developer in producing maintainable software. Consequently, maintenance practices, which maintainers control, and development methodology, which developers control, are candidates for standardization (4). Significant efforts have been made at the National Institute for Standards and Technology (formerly the National Bureau of Standards) to promote standardized maintenance practices through the publication of a series of guides on software maintenance and software maintenance management(5).

The objective of standardization is to improve the maintainability of both existing and new software. However, we should recognize the limitations of using standardization to 'solve' the 'maintenance problem'. These are the following: 1) Much of the software that is maintained was developed without benefit of any methodology; consequently, methodology is of limited use in these cases. 2) Conversely, methodology is most useful when applied to new software. 3) Related to points 1 and 2 is the fact that improvements in maintenance practices are only applicable to existing software. 4) An important determinant of the maintainability of software is the knowledge and skill of the developer and maintainer. 5) There are other aspects of a development methodology, such as expressiveness, that are important when evaluating it as a development tool in addition to its usefulness as an aid for producing maintainable software. Points 4 and 5 are beyond the scope of the paper as are the areas of software engineering environments and tools, which can contribute significantly to the quality of both development and maintenance.

The paper consists of the following sections:

II. Purpose

III. Local Area Network Example

IV. Objective of Maintenance

V. Metrics for Maintenance

VI. Model of Maintenance

VII. Application of Metrics

VIII. Standardization of Change Documentation

IX. Software Communication Mechanisms and Maintenance

X. Standardization through Examination of Development Methodologies

XI. Summary

Each principle of the change management methodology is illustrated by the appropriate part of a local area network (LAN) software maintenance example so that the reader can immediately see an application. The same example is used throughout the paper to maintain continuity for the reader. This example is used to illustrate how mistakes can be made if maintenance is performed without using a formal change procedure. The example is also used to show how mistakes can be avoided by applying the change management methodology. Following the statement of each change methodology principle is an example drawn from the LAN application. The examples are delineated by the use of vertical bars (!).

II. PURPOSE

The first purpose of the paper is to present the case for standardizing software maintenance practices and those aspects of software development methodology that affect the maintainability of the delivered software. The second purpose is to show how to apply the change management methodology.

III. LOCAL AREA NETWORK EXAMPLE

Since the example will be used throughout the paper, it is necessary to present an overview of the LAN software maintenance application at this point. A discussion of all aspects of the changes which were made to the LAN software in the example is beyond the scope of this report.

Figure 1 shows Servers and User Computers on a Token-Ring LAN with the pertinent batch programs keyed to the diagram. Briefly, the functions of these programs are the following:

Server

Autoexec.Bat: Start the Server on network and share resources.

Profile.Bat: Process User Computer identifications to identify configuration (e.g., modem, EGA card, 3270 Emulation, etc.).

Set configuration in memory (the environment).

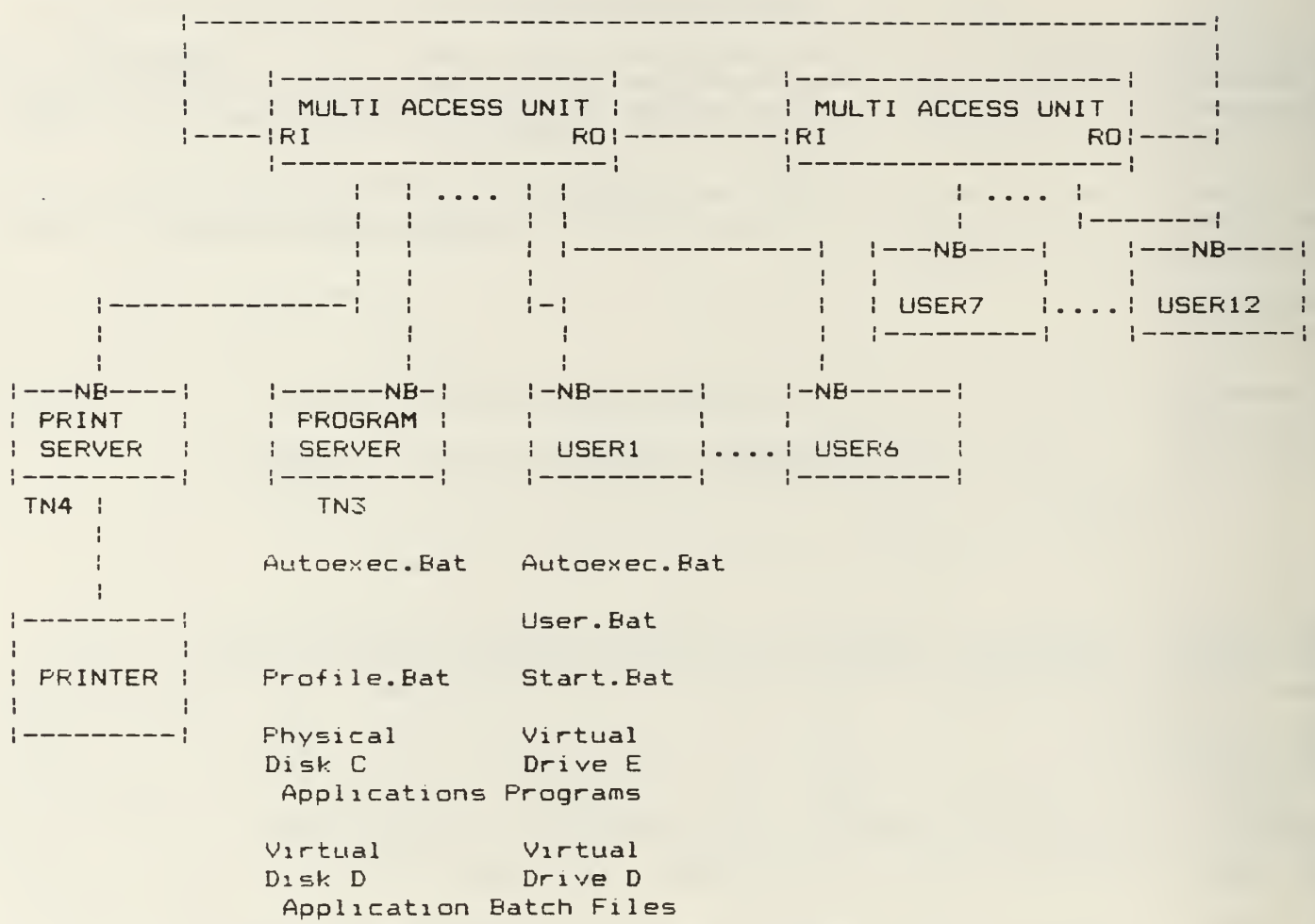
Application Batch Files: Check configuration. Execute application program if required hardware available. Otherwise, display error message.

User Computer

Autoexec.Bat: Set User Computer identification in the environment.

User.Bat : Display logon instructions to the user.

Start.Bat : Start User Computer on network, request resources from Server and call Profile.Bat.



NB: TOKEN-RING BOARD RI: RING IN RO: RING OUT

Figure 1. Token-Ring Network Diagram

A batch (command file) for starting a user personal computer on a local area network (LAN) and requesting resources provided by a server is shown in Figure 2 (Start.Bat) and the corresponding state diagram is shown in Figure 3. This batch file was modified to provide some additional network capabilities as shown in Figure 2; the corresponding modification is shown in Figure 3 with dotted boxes. The boxes represent states and the arrows represent state transitions.

The enhancement provides the ability to store the User Computer configuration (e.g., presence of a modem) in memory and to check the configuration prior to executing an application which requires a given device. The purpose is to prevent the user from wasting time if the required device is not available, and to notify the user of this situation with a message that identifies computers that have the required device. In addition, a significant reduction in software maintenance is achieved by having only one set of application batch files to maintain rather than various sets, with each set tailored to a different configuration. Lastly, this approach achieves a uniform user application program interface.

The numbers on the left side of the commands in the batch file correspond to the numbers on the state boxes on Figure 3. The convention for labeling state transition arrows is: Event/Action. In some cases in Figure 3 there is no event; in these cases 'NE' is used to indicate this. The DOS and PC LAN Program handle transfers of control implicitly (e.g., a transfer of control occurs automatically from PC LAN Program to DOS under certain error conditions). There is no capability in the batch file language for describing error conditions explicitly, although they are shown in the state diagram to clarify the operation.

Asterisks in the batch file identify comments. Unfortunately, the comment concerning accessing the D drive was not changed with the modification. This comment is no longer applicable and caused confusion in trying to understand the program logic. With the modification, neither the D drive nor the directory program 1DIR are accessed at this point in the program. The comment should have been changed to refer to the E drive and the PROFILE program. This affects the transitions from states 5 to 6 and 6 to 7. For the sake of brevity, the error events and actions associated with states 6' and 7' are not shown in Figure 3; they are similar to those for states 6 and 7.

Neither a state diagram nor another type of methodology that would show the consequences of making a change was used in creating the batch program. The use of such a methodology would have helped to avoid this kind of error by:

o Preventing side effects (erroneous comment)

o Providing ability to make selective change (replace commands 6 and 7 with 6' and 7' correctly).

o Identifying existing communication linkages (communication between commands 6 and 7 and the D drive and its directories) and by identifying changed communication linkages (communication between commands 6' and 7' and the E drive and its directories).

```
: *** Start.Bat For Token-Ring User Computer
1 ECHO OFF
: *** Establish Path to Network and DOS Programs Residing on User Computer
2 PATH C:\NETWORK;C:\APPS\DOS
: *** Establish Access for 1DIR Directory Program and its 1DIRDATA
:     Sub Directory
  APPEND C:\1DIRDATA
  ECHO ON
: *** Load Token-Ring Programs
3 TOKREUI
  NETBEUI
: *** Start  the  User Computer on the Network, Using Name Provided by
:     User (Replaceable Parameter %1)  and  Specify  Use  of Resources
:     (e.g., ASG = 10 Devices and Directories)
4 NET START MSG %1 /SRV:1 /ASG:10 /PB1:16K /USN:3 /CMD:12 /SES:18
: *** Request  Use  of  Server Directories on Server TN3 and Printer on TN4:
:     Application  Directory APPS (Virtual Drive E), Application Batch Files
:     DISKD (Virtual Drive D) and Printer PRINT
5 NET USE E: \\TN3\APPS
  NET USE D: \\TN3\DISKD
  NET USE LPT1 \\TN4\PRINT
: *** Access  D  Directory which Contains 1DIR and Application Program
:     Batch Files
6 D:
  *** Load 1DIR
7 1DIR
```

.....

Incorrect
Modification: Replace commands 6 and 7 above with commands 6' and 7':
(comment was not changed)

```
:  *** Access D Drive which Contains 1DIR and Program Batch Files
6' E:
: *** Load Profile
7' PROFILE
```

.....

Correct
Modification: Replace commands 6 and 7 above with commands 6' and 7":
(change comment)

```
: *** Access E Drive, which contains PROFILE program. PROFILE is located
:     on the Server. PROFILE processes User Computer identification  in
:     order to identify the User Computer hardware configuration.    The
:     execution  of  Profile will ultimately lead to the loading of 1DIR
:     and access to application batch files.
6' E:
: *** Load Profile
7' PROFILE
```

Figure 2. Batch file for Token-Ring LAN User Computer Start Program

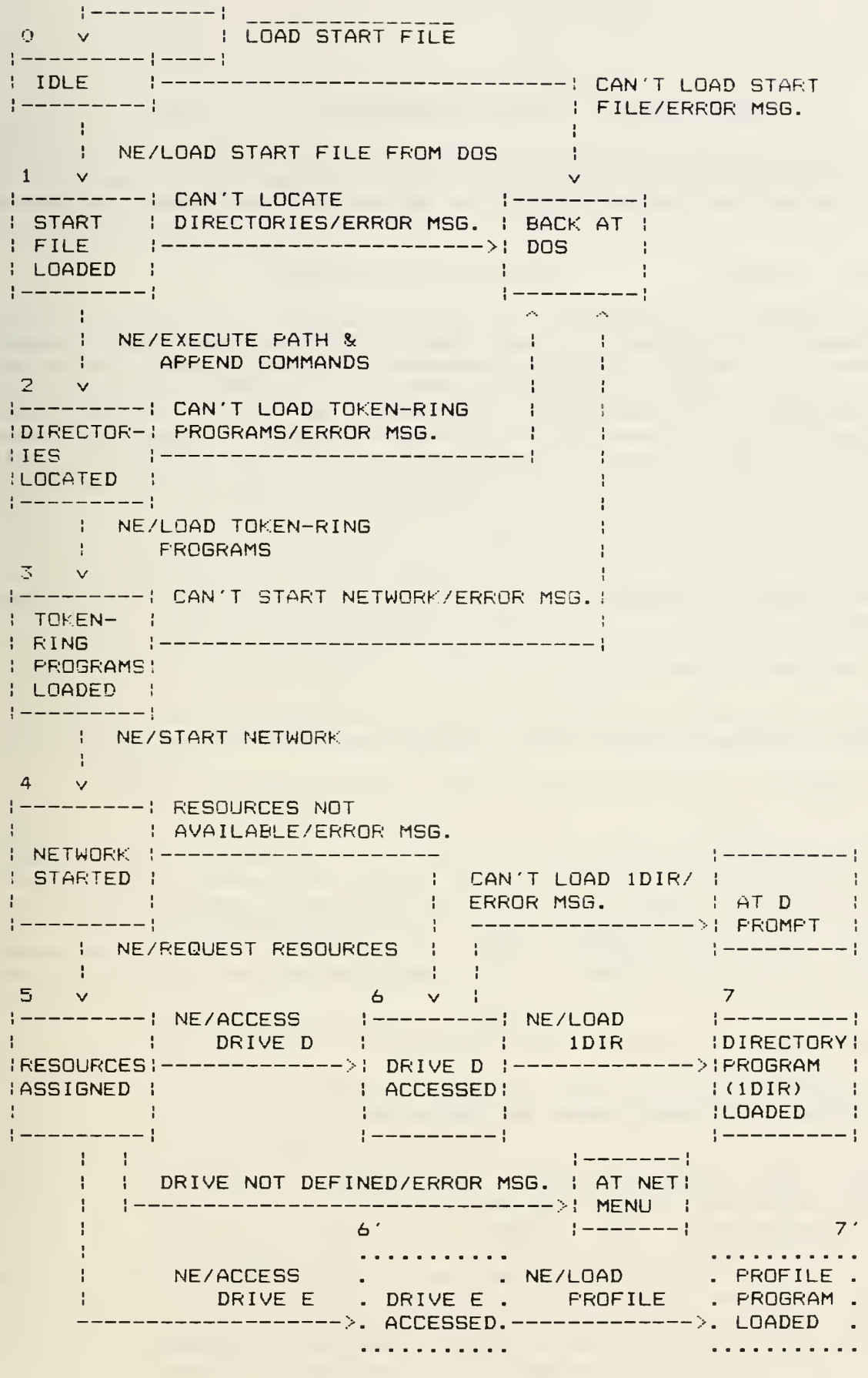


Figure 3. State Diagram of a Token-Ring LAN User Computer Start Program

IV. OBJECTIVE OF MAINTENANCE

The objective of maintenance is to make required changes in software in such a way that its value to users is increased. Required changes can result from either the need to correct errors or to increase the functionality of the software.

A. Maintenance Process

In the broad view of maintenance, it is not limited to making post-delivery changes. Rather, it is a process that starts with user requirements and never ends (6). Even the installation of and changes to a replacement system can be considered part of the maintenance process. Our approach to identifying the maintenance functions which should be standardized is to: 1) Adopt the view that maintenance is a process of change management and 2) Identify tasks in maintenance that are concerned with making changes to software, including changes to documentation (e.g., specification, design, listing, test plan, etc.).

B. Maintenance Tasks

Using the concept of change management, the following maintenance tasks can be identified:

o Identify need for change

¦ The change is desired to prevent users from accessing resources that are not available to them. This will save user time and reduce frustration. ¦

o Determine whether change should be made, based on benefit-cost analysis

¦ The cost is approximately one man-day maximum to code, document and test the change. This amounts to about $ 300 (with employee benefits). This is equivalent to about 100 users saving 5 minutes each, assuming salary of user (with employee benefits) is approximately equal to implementer salary, on the average. The break even point could be achieved within two weeks of implementation, given the number of users and uses of the affected application programs. ¦

Evaluate the effects of change, including possible side effects

o Determine whether change can be made without creating an incompatibility with the rest of the software

¦ The change will not affect user logon instructions. Thus, User.Bat, which contains logon instructions, will not be affected. A change will be required in the user Autoexec.Bat to set the environment (i.e., establish the user computer configuration). This change will have no effect on the user operation. Changes will be required in the application batch files (e.g., Smartcom.Bat) that are stored on the server to add checks of the configuration to see whether the user has the resources necessary to carry out the attempted operation. If this is not done correctly, there will be errors in the operation (e.g., the user will be allowed to attempt an operation that is not possible or will told that the operation is not possible when, in fact, it is possible). ¦

o Make the change, if warranted, and only if it can be done in a standard way

! The change is warranted based on the very favorable cost-benefit relationship. The change can be made in a standard way by using the change management methodology that follows. !

V. METRICS FOR MAINTENANCE

In order to manage software change it is desirable to measure the effects of change. This is accomplished with quality metrics. A quality metric is defined as follows: a quantitative measure of the degree to which software possesses a given attribute that affects its quality (1). Ideally, there would be agreement on a set of application-independent, language-independent, software structure-independent metrics ('universal metrics'). Agreement does not exist in the software engineering community on a universal set. Lacking this agreement, metrics which are known to be related to the effectiveness and efficiency of the software development process are used during development to measure and improve the development process; these are called process metrics (7). It is assumed that their use will result in maintainable software. However, process metrics, like traceability, have little to do with measuring whether the system achieves its quality requirements. For that we need product metrics like reliability, accuracy, response time, throughput, etc. The two types of metrics are related in the sense that high process metric values will contribute to high product metric values. Product metrics are beyond the scope of this report.

The role of metrics in maintenance can be demonstrated by posing the following question:

When a maintenance action is taken, how are the relevant metrics values affected?

o What are the relevant metrics?

! Traceability !

o What were the original values?

! 100 % between code and state diagram !

o What are the new values?

! < 100 % . Can't trace from Start.Bat of User Computer to Server application batch files !

o Examine incremental changes

! Are they in the right direction (e.g., reduced complexity)?

! No. Increased complexity. !

* Are they approximately the right values (e.g., within the bounds of experience with respect to the maintenance action)?

! Traceability will be lost if the change is extensive. The change should not involve more than about 30 % of the batch file. If this is not the case, the batch file should be rewritten (rule of thumb regarding percentage of statements changed). !

VI. MODEL OF MAINTENANCE

To explain the dynamic interaction between development and maintenance, as exemplified by the changes in metrics values that result from development and maintenance actions, the model in Figure 4 is provided. A model of the maintenance process is essential for standardization to be achieved. Different organizations may want to use different metrics, depending on the relevance of the metrics to their maintenance environments and projects.

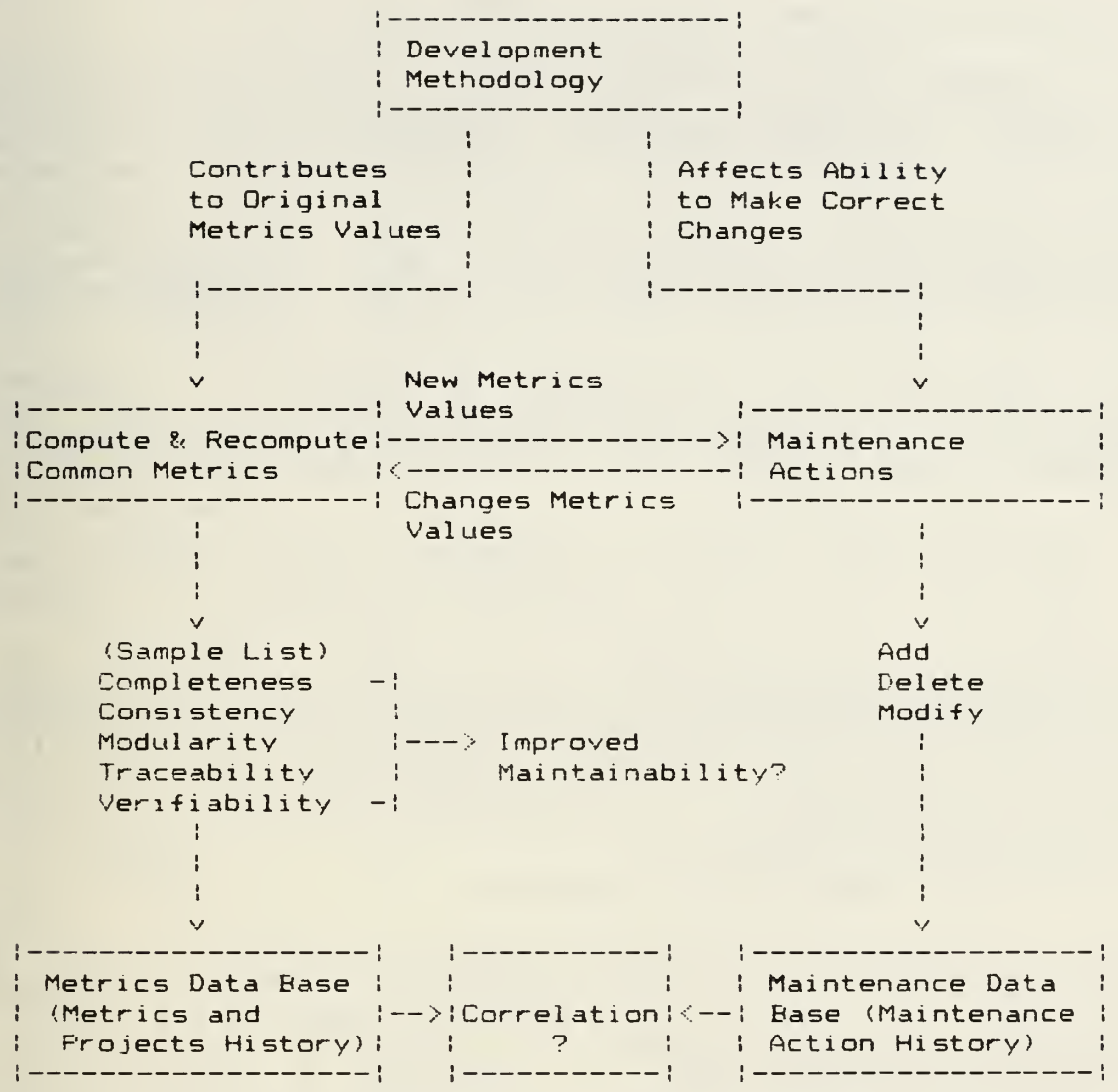


Figure 4. Model of the Interaction between Development, Maintenance and Metrics.

This model may be understood and applied as follows:

A. Evaluate: Estimate the incremental change in metric value of a proposed maintenance action. If the software change is made, measure its effect after the change is made. To the extent feasible, quantify the effect of the change. The following questions are relevant when considering a change to software:

o Given the development methodology and a maintenance action, how will the metrics values be affected (magnitude and sign)? Will they change in a direction to indicate the software will be (or has been) improved? Or will the change indicate that the software will be (or has been) degraded?

This model would assist the maintenance organization to: 1) determine whether a change should be made, 2) determine whether a change improved maintainability, if it was made, and 3) document the history of the project and the change so that this information can be used when making future change decisions.

B. Feedback: Understand that taking a maintenance action changes metrics values and that the new metrics values will influence future maintenance actions.

C. Data bases: Maintain data bases of project characteristics, metrics, and maintenance actions as an aid to learning from the past: Was a given metric a good predictor of the effect of a given maintenance action? Which maintenance actions improved and which degraded the software for given project characteristics? Did the nature of the development methodology influence the maintainability of the software?

VII. APPLICATION OF METRICS

It was mentioned previously that metrics are part of the maintenance model -- they assist in evaluating the effects of change. When used over hundreds of software components (an element of a software system, like a module), the metrics can assume numerical values (e.g., for Completeness: ratio of completed components to total number of components in the system). For a single component, as in the example, a qualitative interpretation is appropriate. This is done below for the example, using typical metrics. Although the modification has improved **functionality**, it has degraded **maintainability**.

TABLE 1

METRICS APPLIED TO EXAMPLE PROGRAM

Metric	Original Program	Modified Program
Completeness:		
Are all required software components present?	Yes	No. The correct comment is missing.
Consistency:		
Are the code and documentation uniform and free of contradiction?	Yes	No. The comment contradicts the commands and vice versa.
Modularity:		
Is the structure cohesive and self-contained?	No	No. Quirks of the DOS language inhibit modularity, but similar commands are grouped.
Traceability:		
Can the program parts be traced from one to another?	Yes	No. Can't trace between commands, drives and directories.
Verifiability:		
Can the correct operation and performance of the program be verified?	Yes	No. The erroneous comment confuses the verification.

VIII. STANDARDIZATION OF CHANGE DOCUMENTATION

Because there is a great difference in applications, programming environments, etc., in various organizations, the maintenance standard should accommodate those differences and specify only a minimum set of requirements and procedures.

Standardization can be viewed as a process of posing questions prior to a maintenance action and having the maintainer answer them. The purpose of this is to ensure that the maintainer has thought about the consequences of proposed changes and is alerted to potential pitfalls. Maintenance decisions and actions should be recorded in a data base for use in making future maintenance decisions.

The entities which are subject to change are software components. For the sake of brevity, 'software component' will hereafter be called 'component'.

A. Documenting the Effects of Change

It should be a standard procedure of maintenance to document a proposed change in the following format (or similar format) and, if the change is made, to fill in as much detail as possible about the change. The items to be considered in deciding on a change are more important than the specific format used to document the change. The Xs in the matrix indicate a relationship between an input item and an output item, and 'DNA' means 'DOES NOT APPLY'.

Change an input (add or modify)

Type: ¦ Batch file statements ¦

Format ¦ PC DOS batch file conventions ¦

Value (How are outliers handled? Are they used or rejected?) ¦ DNA ¦

Range (e.g., extremes of numbers): ¦ DNA ¦

Precision (e.g., number of decimal points): ¦ DNA ¦

Accuracy (e.g., number within X % of actual value): ¦ DNA ¦

Name (Standardize name; should say what component does): ¦ (Start)

Starts User Computer on network and requests resources ¦

Questions:

* What is the effect of input on outputs?

  ¦ Link between Start.Bat on User Computer and Server application batch files ¦

* What is the effect of input on computation of function?

* Computation within bounds? (i.e., does input cause computation to be outside feasible range of numbers in application?):¦ DNA ¦

TABLE 2

EXAMPLE INPUT-OUTPUT CHANGE RELATIONSHIP

			OUTPUT (Name)			
	Type	Format	Value	Range	Precision	Accuracy
INPUT (Name)						
Type	X					
Format		X				
Value			X	X	X	X
Range			X	X	X	X
Precision			X	X	X	X
Accuracy			X	X	X	X

¦ Type: New statements in Start.Bat on User Computer creates need for new statements in application batch files. ¦

¦ Format: If syntax incorrect, won't work. If output not correctly related to input, could make wrong decision about executing application program. ¦

¦ Add/Modify a function or statements: What resources, functions or statements must be present so that change can be utilized? (Need, for example, paths, directories, and disks defined) ¦

B. Documentation Requirements

As a minimum the following should be standard documentation for supporting maintenance: requirements specification, design specification, program listing, test plan, and test results, as summarized below.

Phase	Documentation
Requirements Analysis	Requirements Specification
	¦ Need environment variables to improve useability of LAN and to simplify maintenance. ¦
Design	Design Specification
	¦ State Diagram ¦
Coding	Listing
	¦ Batch File ¦
All	Test Plan, Test Results
	¦ Test use of application batch files from all User Computers.Allow access if configuration permits it; otherwise, disallow it. ¦

IX. SOFTWARE COMMUNICATION MECHANISMS AND MAINTENANCE

Mechanisms which are available for communicating between components are an important aspect of maintenance because of the serious consequences of making an error in adding or changing a linkage. As opposed to other types of software changes, a change in a communication mechanism affects more than one component. This is particularly important for networks where a defective mechanism can adversely affect the operation of computers at remote sites.

A. Kinds of Communication Mechanisms

o Data linkages (for the transfer of data):

- Message passing (can also be control message): ¦ DNA ¦
- Transaction (e.g., update in a data base management system): DNA
- Mail Box (i.e., store data in standard location where it can be used by other processes)

¦ Autoexec.Bat on User Computer stores data (its ID) in standard location (environment) that can be used by Profile.Bat on Server ¦

o Control linkages (for the transfer of control)

- Subroutine call: ¦ DNA ¦

- Procedure call: ¦ DNA ¦

- Remote procedure call (RPC): ¦ DNA ¦

B. Characteristics of Communication Between Software Components

1) Explicit: There is an actual transfer or exchange of data or passing of parameters, or an output from one component is the input to another component, or one component calls another component.

¦ Start.Bat on User Computer calls Profile.Bat on Server ¦

2) Implicit: Based on the position of the given component within a sequence of components (e.g., instructions in a program):

¦ Since PROFILE is needed to determine the User Computer configuration, it is called as the last step ('7) in Figure 2. ¦

3) Indirect: Based on one component providing (e.g., store data in RAM or secondary storage) that another component uses:

¦ Profile.Bat on Server sets configuration environment. Application batch files will reference the environment ¦

Before components are added, deleted or modified, it should be standard procedure to ascertain and document the effects of making the change on inter component communication. Furthermore, if the change is made, as much detail as possible should be documented about the change, as suggested by the questions below.

4) ADD a component

o What other components will the given component communicate with once it is added? ¦ component = batch file statement ¦

¦ Start.Bat on User Computer will communicate with Profile.Bat on Server. The environment will communicate with application batch files on Server ¦

o What are the communication linkages? (parameter passing, message exchange, RPC, etc.?)

¦ Start.Bat on User Computer names an executable batch file (e.g., Profile) and causes the file to be loaded and executed. Autoexec.Bat sets the User Computer identification. Profile.Bat on Server sets the configuration. ¦

o What existing communication linkages will be affected by the change?: ¦ None ¦

5) DELETE a component ¦ DNA ¦

o What communication linkage will be broken by the deletion?

o What are the new communication linkages that result from the deletion?

6) MODIFY a component

¦ Modify Start.Bat and application batch files. These are the components to be modified. ¦

o What is the existing communication linkage which involves this component?

¦ None between Start.Bat and application batch files ¦

o How will this communication linkage be modified by the change in the component?

¦ A communication linkage will be established between User Computer and Server via the environment ¦

X. STANDARDIZATION THROUGH EXAMINATION OF DEVELOPMENT METHODOLOGIES

There is evidence that the characteristics of development methodologies (8) and the characteristics of programming languages (9) can influence maintainability.

A. Characteristics of Development Methodology

When we maintain software we may not be cognizant of the development methodology which was used to produce the software, but it will affect our ability to maintain the software. The evaluation hinges on a single criterion: **does the methodology support the creation of software which is easy to change without inducing side-effects (an unexpected and undesirable result of making a change)?** This objective will be achieved if the methodology forces the designer to formally consider the consequences of making a change once the software has to be maintained (10). It follows that in order to capitalize on a methodology that supports maintenance, it is necessary to use that methodology to maintain the software. The following is a standard procedure for evaluating a methodology with respect to its capability to support maintenance.

Does the methodology (e.g., state diagram) assist to:

1) Identify side-effects when performing maintenance

¦ The state diagram (see Figure 3) can assist in identifying potential side-effects because it shows: changes of state in a program, events that cause changes in state, and resultant actions. For example, in Figure 3 the modified state diagram shows a transition from the Resources Assigned state (step 5) to the 'Drive E Accessed' state (step 6') that could have an effect on loading the directory program (step 7) because the modification has intervened in the original program execution sequence. We could have a problem if this intervention prevents the directory program from eventually being loaded. ¦

2) Provide ability to make selective change (i.e., don't change or destroy another part of the software when making a change)

¦ Obviously, no methodology is foolproof in identifying the consequences of making a change, but a methodology like the state diagram forces the maintainer to consider the effects of change and makes visible the **relationship** between programs. For example, it says in Figure 3 that a program called 'PROFILE' is to be loaded. This raises some interesting questions for the maintainer: What is the program 'PROFILE'? What does it do? Where is it located? The incorrect modification does not answer these questions. The correct modification does. However, notice that the state diagram does not suggest to the maintainer that the comments are incorrect; only the listing can do that. This suggests the impossibility of higher level documentation providing a complete description of program logic. ¦

3) Make **visible** the dependencies between inputs, processes and outputs (dependencies make it difficult to change the software without affecting something else which was working correctly prior to the change)

¦ Dependencies are created by the change between Start.Bat and application batch files and between Start.Bat and PROFILE. These are unavoidable, given the approach for checking User Computer configuration. However, the state diagram helps to make these dependencies visible. They would be more visible to the reader if the state diagram for the processing of the PROFILE program were shown (not shown because it is beyond the scope of this report). ¦

4) Determine whether change can be made without creating an incompatibility with the rest of the software

¦ This would be determined by analyzing the application batch files and state diagrams to see whether an incompatibility in their operation would be created by checking the User Computer configuration. ¦

5) Support a rational change policy:

- o Make a change, if warranted, and only if it can be done in a standard way, a 'standard way' being defined as being in conformance with the above procedure for assessing the impact of change.

  ¦ To assess the effect of the change, we determine how the changed Start.Bat will affect the application batch files.

  There are three possibilities:

  - They will no longer work at all
  - They will deny program access when the required equipment is available
  - They will allow program access when the required equipment is not available
  - They will work satisfactorily ¦

- o Make changes in small, controlled increments

  ¦ By breaking the logic into discrete state transitions (e.g., transition from 'RESOURCES ASSIGNED' (Step 5) to DRIVE E ACCESSED (Step 6) in Figure 3), changes are kept small. Also the individual changes are kept small by distributing parts of the total change to several batch files (e.g., Start.Bat, Profile.Bat and application batch files). However, the incorrect modification in Figure 2 is uncontrolled, raising questions about the function and location of PROFILE and its relationship to Start.Bat. ¦

B. Characteristics of Programming Language

Characteristics of the programming language can also significantly influence the ability to maintain (9). Two brief examples from the DOS language will be given:

o PATH command: If this command appears once and is repeated, the most recent occurrence of the command is the only one in effect. This means that any paths used to establish directories in a previous occurrence are lost unless they are repeated in the new PATH command. In effect, this means that a new path must be a superset of the previous path, if all original directory information is to be retained. However, this could result in long path commands and, without writing complicated logic, commands are limited to a single line! Thus the maintenance principle of being able to make a selective change (i.e., one wants to just add or delete parts of the PATH command, not write a new one) cannot be achieved with this command.

o IF command: The IF command has the format: IF string1==string2 command. The requirement for the second '=' is unexpected. This nuance of the language has caused us to make several errors in writing network batch files. This seemingly minor item can cause havoc in maintenance because a frequent change to batch files occurs as the result of adding capabilities to the network that are conditioned on the availability of certain resources. The IF command is key to specifying these conditions.

XI. SUMMARY

We have proposed that maintenance can be improved through standardization. The elements of the proposed standardization process are the following:

o Metrics

o Model of maintenance

o Change documentation

o Software communication mechanisms

o Development methodology supportive of maintenance

An example was presented of the application of one development methodology -- state diagrams -- to illustrate how proposed and accomplished changes can be illuminated so that errors can be avoided and maintainability improved.

Various methodologies could have been used to illustrate the change management methodology. What is important is not the particular development methodology, but the **consistent** application of a selected methodology, using the change management methodology which has been described. Additional research is needed to test other types of applications, programming languages and changes against the change management methodology.

XII. REFERENCES

(1) An American National Standard IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Standard 729, 1983.

(2) Bennet P. Lientz and E. Burton Swanson, "Problems in Application Software Maintenance", Comm. ACM, vol. 24, no. 11, pp. 763-769, Nov. 1981.

(3) Bennet P. Lientz and E. Burton Swanson, "Software Maintenance Management", Reading, MA: Addison-Wesley Publishing Co., 1980.

(4) Norman F. Schneidewind, "Software Maintenance: The Need for Standardization", Proceedings of the IEEE, April 1989

(5) James A. McCall, Mary A. Herndon, and Wilma Osborne, Software Maintenance Management, National Bureau of Standards Special Publication 500-129, October 1985.

(6) Meir M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution", Proc. of the IEEE, vol. 68, no. 9, pp. 1060-1076, September 1980.

(7) Rome Air Development Center, RADC-TR-85-37, Final Technical Report, February 1985.

(8) Bob Britcher and Jim Craig, "Upgrading Aging Software Systems Using Modern Software Engineering Practices: IBM-FSD's Conversion of FAA's National Airspace (NAS) En Route Stage A Software From 9020s to S/370 Processors', Proceedings, Conference on Software Maintenance-1985, Computer Society Press, pp. 162-170.

(9) Grady Booch, Software Engineering with Ada, The Benjamin/Cummings Publishing Company, Inc., 1983.

(10) Harlan D. Mills. "Stepwise Refinement and Verification in Box-Structured Systems", Computer, vol. 21, no. 6, June 1988, pp. 23-36.

DISTRIBUTION LIST

Mr. Roger Daugherty, Code 01B Navy Management Systems Support Office Naval Air Station Building R52-7 Norfolk, VA 23511-6694	1
Commanding Officer Navy Management Systems Support Office Naval Air Station Norfolk, VA 23511-6694	1
Technical Director Navy Management Systems Support Office Naval Air Station Norfolk, VA 23511-6694	1
Prof. Tarek Abdel-Hamid Code 54Ah Naval Postgraduate School Monterey, CA 93943	1
Prof. Norman Schneidewind Code 54Ss Naval Postgraduate School Monterey, CA 93943	10
National Technical Information Center Cameron Station Alexandria, VA 23314	2
Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Computer Center Library Code 0141 Naval Postgraduate School Monterey, CA 93943	1
Administrative Sciences Department Library Code 54 Naval Postgraduate School Monterey, CA 93943	1
Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943	1